

Computer Science Department

TECHNICAL REPORT

Optimal Distributed Resource Allocation

by

John Reif*

and

Paul Spirakis

Technical Report No. 62

March 1983

NEW YORK UNIVERSITY



Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012

CSD TR-67



Optimal Distributed Resource Allocation

by

John Reif*

and

Paul Spirakis

Technical Report No. 62

March 1983

This work was supported in part by the National Science Foundation Grant NSF-MCS79-21024, and the Office of Naval Research Contract N000 14-80-C-0647.

* Aiken Computation Laboratory, Harvard University.

c 1

Optimal distributed resource allocation

by

John Reif* and Paul Spirakis**

*Aiken Computation Laboratory, Harvard University

**Courant Institute of Mathematical Sciences

New York University

This work was supported in part by the National Science Foundation Grant NSF-MCS79-21024, and the Office of Naval Research Contract N00014-80-C-0647.

Abstract

In this paper we consider a resource allocation problem in a distributed network, where processors' speeds can vary dynamically. The problem is local in the sense that the maximum number of users competing for a particular resource at any time instant is upper bounded by a constant v and also at any time instant the maximum number of resources that a user is willing to get is upper bounded by a constant k . Each user's demands for resources may change dynamically. Our solutions to the problem are real time since our algorithm's response time is upper bounded by a function which does not depend on any global measure of the system of processes and resources, except k and v .

We improve here previous real time solutions given to the problem in [Reif, Spirakis, 1982B], which had response time of $O(v^k)$. In this paper we provide probabilistic algorithms whose response time is polynomial to v and k . Our most efficient algorithm has response time $O(v \cdot k)$.

Unlike the probabilistic algorithms of [Reif, Spirakis, 1982B], we do not need to use random waits as means of avoidance of process starvation and of achieving probabilistic fairness. Instead, our algorithm employs a new method of probabilistic bidding, to resolve contention of users for resources. This technique is essential in achieving polynomial response time.

We furthermore prove that our solution is optimal with respect to the average response to a user's request. In particular we provide matching lower bounds for any distributed algorithm, and these bounds are within a constant factor of the response time of our own algorithms. We also provide a suboptimal algorithm which is useful for improving the throughput rate of resource allocations in systems where the majority of users is willing to get only a few resources. This suboptimal algorithm does use random waits combined with probabilistic bidding.

Our techniques employ limited parallelism within each process, together with the probabilistic bidding. (This limited parallelism is useful in achieving optimal response time, though we would still get polynomial response time without limited parallelism).

1. Introduction

1.1 Resource Granting Systems

In this paper, we consider a resource allocation problem similar to that described in [Lynch, 1980] and [Reif, Spirakis, 1982B]. There are potentially an infinite number of processes in the system, and each has an integer name. Let the set of processes be called Π . There is a potentially infinite set of resources, p , in the system. Let $R \subseteq \Pi$ be the set of processes which control resources. Let U (the set of user's processes) be $\Pi - R$. Each resource $p(j) \in p$ is controlled by a distinct granting process j . This process has the responsibility of granting its resource to a distinct user process in U , i.e. no two different processes in U may use the same resource at the same time. We assume user processes communicate only with those granting processes for which they request resources.

A system as above is called Resource Granting System (RGS); see [Reif, Spirakis, 1982B]. The goal of an RGS is to satisfy dynamically changing user requests for resource allocation, in a distributed way, by only a local communication between granting and requesting processes. We assume that there is a global time t totally ordering events, but processes may not have access to the global time.

We now describe the adverse oracle A which has the power to set actions in the worst possible way to increase the response time. A has also the capability to select at time $t=0$ the schedule of the speeds of all processes at all times $t > 0$. In addition, at each time $t > 0$, the requests by user processes U are specified by A . The adverse oracle A is restricted, to allow users to keep asking for their resources until they are granted. In practice, no such A may exist but coincidence of worst case situations may replace its action.

A process step consists of either an assignment of a variable, a test, a logical or arithmetic operator or a no-op. A step is considered to be a finite time interval Δ in which a single primitive instruction is executed just at the last moment of Δ , and no other instructions are executed within Δ . Let a process be tame during a time interval Δ , if for any interval Δ' which intersects Δ and is a single step of the process, then $|\Delta'| \in [r_{\min}, r_{\max}]$ where $r_{\min}, r_{\max}$ are

fixed real constants with $0 < r_{\min} < r_{\max}$. We will not require processes to be tame at all times. However our proof that our algorithms are real time makes an assumption that processes are tame. We assume that processes are reliable in the sense that they perfectly execute their programs. The rate of execution varies dynamically. We require that, at no time, any granting process $i \in R$ simultaneously grant the resource $p(i)$ to more than one requesting process. We also require that, as soon as a process $j \in U$ has got all its required resources, then it can keep them only for a time interval whose length is upper bounded by a fixed parameter δ (containing at most an upper bounded number of its steps, $\mu = \delta / r_{\min}$, if the process is tame). Let $resources_t(i)$ be the set of resources that process i is requesting at time instant t . Let $k_{i,t} = |resources_t(i)|$. Let $askers_t(j)$ be the set of user processes requesting $p(j)$ for $j \in R$ at time t . Let $v_{j,t}$ be $|askers_t(j)|$. We assume that at all times $t \geq 0$, $v_{j,t}$ and $k_{i,t}$ are upper bounded by constants v, k respectively and that $k < v$. (This does not necessarily imply any bounds on $\sum_{t=0}^{\infty} |resources_t(i)|$ or $\sum_{t=0}^{\infty} |askers_t(j)|$ for any $i \in U$ or $j \in R$).

With respect to interprocess communication we assume (1) that each resource allocator $j \in R$ has a set S_j available to it of size at most v , containing the names of those processes willing to get the resource. As in [Lynch, 1980] we assume this to be a primitive of our system (it could be implemented by a queued message system, for example) (2) that synchronization must be done by special variables, called flags, each of which is written only by one process and read by at most one other process. Read-write conflicts on flags are excluded by our process step semantics and by our notion of global time. Flags seem to be the simplest primitives for synchronization and lead to an easy implementation (in contrast to e.g. distributed shared variables of multiple readers and writers).

1.2 Implementations and Complexity of an RGS.

An implementation of an RGS determines the synchronization algorithms that the processes run. As stated above, the synchronization algorithms use only flags to implement the synchronization between processes. We consider a time-varying

hypergraph H_t with node set Π and time-varying hyperedge set $E_t = \{[i]$ union resource $_t(i): i \in U\}$, i.e. where a hyperedge at time instant t is the set of nodes of Π consisting of a single process i in U and the granting processes of the resources i is willing to get at t . An RGS implementation dynamically achieves distributed matchings in the hypergraph H_t .

For each adverse oracle A , let the response time of the RGS implementation be the random variable $\gamma_{A,k}$ which is the length of the smallest interval Δ required for any process $i \in U$ to have k resource requests simultaneously granted, given that i requested these resources during the entire interval Δ and assuming that, i and all allocators of the resources requested by i within Δ , are tame within Δ .

Let the mean response $\bar{\gamma}_k$ be the $\max\{\text{mean}\{\gamma_{A,k}\} \text{ over all oracles } A\}$. Let the error response be the minimum $\gamma_k(\epsilon)$ such that for every oracle A

$$\text{prob}\{\gamma_{A,k} < \gamma_k(\epsilon)\} > 1 - \epsilon.$$

The RGS implementation is real time if $\forall \epsilon \in (0,1], \gamma_k(\epsilon) > 0$ and upper bounded by a function independent of any global measure of the network. (Note: A global measure of the network is any positive function g of $h = |\Pi|$ such that $\lim_{h \rightarrow \infty} \frac{v}{g(h)} = 0$). Note that if an RGS implementation is real time, then the mean response $\bar{\gamma}_k$ is also upper bounded by a function independent of any global measure of the network.

1.3 Previous work

[Rabin, 1980a] first applied probabilistic choice to synchronization problems in distributed systems and provided a solution to the dining philosophers problem which, with probability 1, is deadlock free and starvation free. [Rabin, 1980b] applied probabilistic coordination methods to synchronize access of processes to a critical resource in a space-efficient manner.

[Lynch, 1980] first posed the localized resource allocation problem as a formal synchronization problem. Let the resource graph G be the graph whose nodes are the resources and two resources are

connected by an edge if there is ever a user process requesting both of them, maybe at different times. Let $x(G)$ be the chromatic number of G . The implementation proposed by Lynch was a deterministic one in which processes should know the color of each resource in a coloration of G . The response time achieved in [Lynch, 1980] is of the order of $x(G)v^{x(G)}\tau$ where τ is the time necessary for interprocess communication. This was not a real time implementation since $x(G)$ could be $\Theta(|\Pi|)$ in general.

[Reif, Spirakis, 1982B,] provided the first real time probabilistic RGS implementation of $\gamma_k(\epsilon) = O(kv^k \log(\frac{1}{\epsilon}) \log(\frac{v}{\epsilon}))$ and $\bar{\gamma} = O(kv^{k+2} \log v)$. In that previous paper, we used the techniques of probabilistic selection of processes by resource allocators and random waits to avoid adverse schedules of speeds which might be set up by the oracle. Although this was ~~not a~~ real time implementation, it is still exponential in k .

1.4 The new results of this paper.

We shall present in section II a probabilistic implementation of an RGS, with mean response $\bar{\gamma}_k = O(kv)$ and $\gamma_k(\epsilon) = O(kv \log(\frac{1}{\epsilon}))$. To achieve this response, we make essential use of a new probabilistic bidding technique, together with use of limited parallelism within each user process and each resource allocator. In our primary algorithm we do not use random waits to achieve probabilistic fairness. Instead we use only the new probabilistic bidding technique. In particular, we slice time of each process into rounds. In each round each user process tries to get all the wanted resources. It has to get all of them in the same round. The users deny the allocation of resources to them, unless all the required resources are offered to be allocated in a small number of steps. At the end of the round, users release their allocated resources (if any) and make a fresh start. User rounds have the same length in steps for all users and this length is a parameter of the algorithm. In contrast, resource rounds are not of the same length and their length in steps is not fixed in advance, but adjusts to the conditions of the algorithm. We conjecture that this is essential in avoiding exponential growth of the response with k .

We also prove lower bounds of $\Omega(kv)$ for the worst case and average

response time of any algorithm for the stated local resource allocation problem, thus our proposed technique is of optimal performance within a constant factor.

Finally, we provide a non-optimal algorithm which is of polynomial mean response time in k and is useful for improving the throughput rate of resource allocations in the network. In particular, it allows user processes which demand less than k resources to have higher probability of being assigned. This algorithm has the property that if a user $i \in U$ has a request of at most $k_i \leq k$ resources, then it has mean response $\bar{Y}_{k_i} = O(v \cdot k_i \cdot \log k \cdot \log(k_i v))$.

1.5 Applications

Example 1: k-fork Philosophers

As a simple example of an RGS we consider a generalization of the dining philosophers problem to the case where each philosopher requires k forks to eat. Note that we can extend this to the case where the identities of the forks required by each philosopher change dynamically. Let the set of "forks" be $R = \{r_1, \dots, r_n\}$ and the set of "philosophers" be $U = \{u_1, \dots, u_n\}$ and let

$$\text{resources}_t(u_i) = \{r_i, r_{(i+1) \bmod n}, \dots, r_{(i+k-1) \bmod n}\}$$

and

$$\text{askers}_t(r_i) = \{u_{(i-k+1) \bmod n}, \dots, u_{(i-1) \bmod n}, u_i\},$$

for all t . Our new resource allocation algorithm achieves mean response time $\bar{Y}_k = O(k^2)$. In contrast, our previous results achieved mean response time $O(k^{k+3})$.

Example 2 Two-phase locking in databases

Two-phase locking is a concurrency control method in databases; for a survey see [Bernstein, Goodman, 1980]. It has the feature that as soon as a transaction releases a lock, it never obtains additional locks. A very efficient static implementation of two phase locking can

be achieved by our methods. Our assumption is that transactions are allowed to act on the data only if they got all the locks requested. In the context of such a database system, let the users in U be called transaction modules and the processes of R be called data modules. If the readsets of the transactions are of cardinality at most k at each time instant and if at most v transactions can compete for a lock at a time instant t , then our optimal RGS will result in an $O(vk)$ mean response time per transaction. Our suboptimal RGS achieves an even smaller mean response time when $|\text{readset}_i(t)| = o(k)$. In this case, if a transaction wants to lock k_i data items at a time, it has a mean response of $O(vk_i \log(vk_i) \log k)$ time. (However this becomes $O(vk \log(vk) \log k)$ when $|\text{readset}_i(t)| = k$). Our implementations of two phase locking proposed in this paper are asymptotically much more efficient than the static locking method proposed in [Reif, Spirakis, 1982B], which had a mean response time $O(kv^2)$.

2. Our distributed probabilistic bidding algorithm.

We assume that the requesting processes communicate only to the resource allocators whose resources they want (or have been allocated), and that granting processes j are willing to communicate only to the requesting processes in the set S_j (as defined in Section 1.1). The actions of the requesting and granting processes are time-sliced in rounds, each round being a repetition of a basic set of actions. Processes use independent sequences of probabilistic choices as the basic counteract to adverse speed schedules and adverse resource demands set up by the oracle A . We assume that A cannot affect or foresee the results of these probabilistic choices. We allow each user in V and each resource allocator in R to have a set of synchronous parallel subprocesses, which aid in our algorithms. The use of local parallelism here is not actually essential in achievement of polynomial response time.

2.1 An informal description of the rounds.

(a) The user's round

A user's round starts with the user drawing (with equal probability) a random number in the set $\{1, 2, \dots, \beta kv\}$ where $\beta > 1$ is an integer. If the number drawn was less than kv , the user remains inactive, until the end of the rounds. (All users' rounds take a predetermined number of steps). Else, the user immediately notifies (by the use of at most k parallel synchronous subprocesses) all the resource allocators of the resources he wants, that he is a winner. Then, the user's parallel subprocesses collect answers from the resources for a period which is bounded by a constant number of steps. During this period some of the resources may declare that they agree to be allocated to the particular user. However, if at that time, any other resource requested by that user is denied, then that user does not utilize the resources which agreed to be allocated to him, but he continues to report that he is a winner to all of his requested resources and repeats the algorithm (without drawing again), until the user's round ends. If all of the wanted resources agree to be allocated at the same time, the user utilizes them for μ steps and then he releases these resources. This is done in parallel, by explicitly notifying the granting processes of the release, using his k subprocesses. Note that a communication with all the k resource allocators takes only $r_{\max}$ time due to the limited parallelism and tameness of processes.

(b) The resource allocator's round.

A resource allocator's (j) round starts with a monitoring period (of a constant number of steps) during which at most v parallel synchronous subprocesses continuously monitor the users of the set S_j , looking for winners. Let M_j be the set of winners detected during the monitoring period. If M_j contains more than one winner, then all the elements of M_j are notified (in parallel) that they have been denied, and the round ends. However, if M_j has a unique winner, then the granting process notifies the winner that it agrees to be allocated. If the winner does not accept the agreement then the round ends. If the winner accepts, then the round enters an allocation period. During

this period, the parallel subprocesses of the resource allocator deny all appearing winners. The round now ends by receipt of the notification by the user that the resource has been released.

Note that communication with all v of the user processes and all set operations in a resource allocator's round take only a constant (independent of v or k) number of steps due to the parallelism employed. Note also that the following hold with certainty:

A resource decides to be allocated to a unique winner, only after the resource allocator agrees to allocate the resource and the winner accepts the agreement. Thus, no resource can be allocated to more than one user at the same time by our bidding algorithm.

For a detailed description of the algorithm, see appendix I.

2.2 Properties of the Bidding algorithm.

Assuming all processes terminate, we shall prove that

Proposition 1. If a particular user i is a winner in its current round, then its k parallel synchronous subprocesses will, at least once in its current round, report at the same time that i is a winner when all requested resources are in a monitoring period.

Proposition 2. Given that a particular user i is a winner in its current round, the probability that he stays a unique winner for all his wanted resources during the whole round, is lower bounded by a constant, independent of k or v .

Let a draw by a user be a random independent selection of one of the numbers $\{1, 2, \dots, \beta kv\}$, $\beta > 1$.

Proposition 3. During a round of user i , any other distinct user j competing for the same resource cannot draw for more than $\frac{r_{\max}}{r_{\min}}$ times.

These propositions will help us prove

Theorem 1. The probability that a user is allocated all his wanted resources in his current round is upper bounded by $\frac{1}{\beta kv}$ and lower bounded by $\frac{1}{dkv}$ where $d > 1$ is a constant, depending on $\frac{r_{\max}}{r_{\min}}$ and independent of k or v .

Theorem 2. $\gamma_k(\epsilon) = O(kv \log(\frac{1}{\epsilon}))$ and $\bar{\gamma}_k = O(kv)$ for the bidding algorithm.

For proofs see Appendix II.

A corollary of theorems 1 and 2 is that our algorithms do not deadlock, no process starves (these hold with probability 1) and our algorithms are probabilistically fair.

3. An $\Omega(kv)$ lower bound for the local resource allocation problem.

Theorem 3. For $k > 0$, there is a network in which at least one user process has to have a response time of at least $(kv-1)\mu$ steps.

Proof

Consider a network with a set of resources R such that $|R| = k$ and a set of users U such that $|U| = kv$. Let A be an oracle such that all processes are equispeed, synchronous and such that $\forall t > 0, \forall j \in U, |\text{resources}_t(j)| > (k/2) + 1$. Then, only one user process can be granted all its resources at each time instant t . To see this, assume for sake of contradiction that at least two users j_1, j_2 are granted all their resources at time t . Each resource has to be granted to only one user at a time, thus forming a bipartite matching of the hypergraph H_t , as defined in Section 1.2. Hence we conclude that the number of allocated resources at time t is

$$|\text{resources}_t(j_1)| + |\text{resources}_t(j_2)| > k+2 > k \quad \text{for all } j > 0.$$

This implies that resources will be allocated to processors serially, hence the last process of the serial order will have a response time of at least $(kv-1)\mu$ steps. The above holds independently of the synchronization technique.

Corollary. Our probabilistic bidding algorithm of Section 2 has optimal mean response within a constant factor.

Proof. By Theorem 3 and by the fact that given any multiset of serial orders of kv elements, there is at least one element whose average position (over the multiset of orders) is at least $\lceil kv/2 \rceil$.

4. An alternative distributed resource allocation algorithm

which improves the throughput

4.1 Motivation

Theorem 3 provided lower bounds for systems which are saturated with requests. In practice, systems will not continuously have so many requests at all times. This section provides an algorithm which gives good response time in the case in which the system is not saturated, (though it is not asymptotically efficient for saturated systems, as the algorithm of Section II).

4.2 Our alternative distributed allocation algorithm

The user process's rounds are the same as in Section 2.1, with one addition: Each round now starts with the user waiting (nonactive) for a randomly chosen number of steps, uniform in an interval upper bounded by a suitable constant c_1 , chosen in such a way that $c_1 \cdot r_{\min}$ is greater than the maximum possible duration of the useful part of the round. The resource allocator's rounds are split into a series of $\log k$ intervals. For each $m = 0, \dots, \log k$, in each interval Δ_m only the users j for which $k_j \in [\frac{k}{2^{m+1}}, \frac{k}{2^m}]$ are monitored. Each resource allocator proceeds to the next interval Δ_{m+1} only if all user processes which demand k_j resources, k_j in $[\frac{k}{2^{m+1}}, \frac{k}{2^m}]$, have been allocated their resources. Within each interval, the resource allocators go through a sequence of rounds, as described in Section 2.1.

4.3 Probabilistic Analysis

Let us consider a time interval Δ_m . Let u' be the number of rounds required for user i with k_i in $[\frac{k}{2^{m+1}}, \frac{k}{2^m}]$, and also for all (at most $k_1 v$) users competing with user i , to have all resources allocated. Let u be the number of rounds (within interval m) required just for user i . Let $u(\epsilon')$ be such that $\text{Prob}\{u < u(\epsilon')\} > 1 - \epsilon'$. Then $\text{Prob}\{u' < u(\epsilon')\} > (1 - \epsilon')^{k_1 v}$, because users take independent actions and due to the random waits. Let us set $\epsilon' = \epsilon / k_1 v$.

Then, we get

$$\text{Prob}\{u' < u(\frac{\epsilon}{k_i v})\} > (1 - \frac{\epsilon}{k_i v})^{k_i v} > 1 - \epsilon$$

Hence, the ϵ -error response for all users in the interval Δ_m , is

$$\gamma_k(\epsilon) = O(h_m v \log(\frac{h_m v}{\epsilon}))$$

where $h_m = \frac{k}{2^m}$, leading to a mean response $\bar{\gamma}_{h_m} = O(h_m v \log(h_m v))$ for the interval Δ_m . The above determines probabilistic upper bounds on the ϵ -error length and mean length of the interval Δ_m . Since there are $\log k$ intervals, the ϵ -error response for a user with k_i in $[\frac{k}{2^{m+1}}, \frac{k}{2^m}]$ will be

$$\gamma_{k_i}(\epsilon) = O(\frac{k}{2^m} v \log(\frac{kv}{2^m \epsilon}) \log k) \text{ implying}$$

a mean response $\bar{\gamma}_{k_i} = O(\frac{kv}{2^m} \log \log(\frac{kv}{2^m}))$. For users with small demands the above mean response outperforms the probabilistic bidding of Section 2. E.g., for k_i in $[\frac{k}{2^{m+1}}, \frac{k}{2^m}]$ and $2^m = \theta(k)$, we get a mean response $\bar{\gamma}_{k_i} = O(v \log k \log v)$.

Hence we have

Theorem 4. Our alternative distributed resource allocation method of Section 4, has the following property: For users i with demands k_i in $[h_m/2, h_m]$ and $h_m = 2^{-m} \cdot k$, the ϵ -error response

is $\gamma_{h_m}(\epsilon) = O(h_m v \log \log(\frac{h_m v}{\epsilon}))$ and the mean response is $\bar{\gamma}_{h_m} = O(h_m v \log k \log(h_m v))$.

References

- Andrews, G., "Synchronizing Resources," ACM Trans. on Programming Languages and Systems, Vol. 3. No. 4, Oct. 81, pp. 405-430.
- Angluin, D., "Local and Global Properties in Networks of Processors," 12th Annual Symp. on Theory of Computing, Los Angeles, CA April 1980, pp. 82-93.
- Arjomandi, E., M. Fischer, and N. Lynch, "A Difference in Efficiency Between Synchronous and Asynchronous Systems," 13th Ann. Symp. on Theory of Computing, April 1981.
- Bernstein, A.J., "Output Guards and Nondeterminism in Communicating Sequential Processes," ACM Trans. on Prog. Lang. and Systems, Vol. 2, No. 2, April 1980, pp. 234-238.
- Berstein, P. and N. Goodman, "Fundamental Algorithms for Concurrency Control in Distributed Database Systems," CCA TR. Contract No. F30603-79-0191, Cambridge, MA, 1980.
- Dennis, J.B. and D.P. Misunas, "A Preliminary Architecture for a Basic Dataflow Processor," Proc. of the 2nd Annual Symp. on Computer Architecture, ACM IEEE, 1974, pp. 126-132.
- Fisher, M.J., N.A. Lynch, J.E. Burns, and A. Borodin, "Resource Allocation with Immunity to Limited Process Failure," 19th FOCS, 1979, pp. 234-254.
- Frances, N. and Rodeh, "A Distributed Data Type Implemented by a Probabilistic Communication Scheme," 21st Ann. Symp. on Foundations of Computer Science, Syracuse, New York, Oct. 1980, pp. 373-379.

- Hoare, C.A.R., "Communicating Sequential Processes," Com. of ACM, Vol. 21, No. 8, Aug. 1978, pp. 666-677.
- Lehmann, D. and M. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers' Problem," to appear in 8th ACM Symp. on Principles of Program Lang. Jan. 1981.
- Lipton, R. and F.G. Sayward, "Response Time of Parallel Programs," Research Report #108, Dept. Comp. Science, Yale Univ., June 1977.
- Lynch, M.A., "Fast Allocation of Nearby Resources in a Distributed System," 12th Ann. Symp. in Theory of Computing, Los Angeles, CA April 1980, pp. 70-81.
- Rabin, M., "N-Process Synchronization by a $4 \log_2 M$ -valued Shared Variable," 21st Ann. Symp. on Foundations of Comp. Science, Syracuse, NY, Oct. 1980. pp. 407-410.
- Rabin, N., "The Choice Coordination Problem" Mem. No. UCB/ERL M80/38, Electr. Research Lab., Univ. of California, Berkeley, Aug. 1980.
- Reif, J.H., and P. Spirakis, "Distributed Algorithms for Synchronizing Interprocess Communication Within Real Time," 13th Ann. ACM Symp. on Theory of Computation, Wis. 1981, pp. 133-145, also rewritten as "Real-time Synchronization of Interprocess Communications," TR-23-80, Aiken Comp. Lab., Harvard University, Cambridge, MA.
- Reif, J.H., and P. Spirakis, "Unbounded Speed Variability in Distributed Communications Systems," 9th ACM Symp. on Principles of Programming Languages, January 25-27, 1982A, Albuquerque, New Mexico.

Reif, J.H. and P. Spirakis, "Real Time Resource Allocation in Distributed Systems" ACM SigAct-Sigops Symp. on Principles of Distributed Computing, August 1982B, Ottawa, Canada.

Schwarz, J., Distributed Synchronization of Communicating Sequential Processes," DAI Research Report No. 56, Univ. of Edinburg, 1980.

Appendix I

A detailed description of the probabilistic bidding algorithm.

In the following, we set parameters $\beta = 2 \frac{r_{\max}}{r_{\min}}$,
 $c = (2(2+\mu)+1) \frac{r_{\max}}{r_{\min}}$. (Only for the suboptimal alternative

algorithm of Section 4, $c_1 = (2(2+\mu)+1) \left(\frac{r_{\max}}{r_{\min}}\right)^2$.

The users use the following flags: For user i the flag $W_{ij} = 1$ iff i is a winner and is willing to get resource $p(j)$. The flag $A_{ij} = 1$ iff user i accepts the allocation of the resource $p(j)$. Both flags are 0 else. The flag N_{ij} is usually 0, it becomes 1 when user i releases resource j .

The resource allocators j use the following flags: $E_{ji} = 0$ if the resource is denied and 1 if j agrees that its resource is allocated to i . Each allocator j has also a shared (for its parallel subprocesses) variable M_j which allows concurrent reads, and, in case of multiple writes of the same value, their sum is recorded. M_j is used to count winners during the monitoring period. Each user i uses a shared (for all its parallel subprocesses) variable L_i . It allows concurrent reads and concurrent writes of the same value. In case of concurrent writes of different values, the smaller value is recorded. L_i is used to identify situations in which all wanted resources have been proposed to be allocated to user i , at the same time.

The counters counter_i , counter_j , count steps of i, j in a round.

The user's round for user i .

(Initially $W_{ij} = A_{ij} = N_{ij} = 0 \forall j = 1, \dots, k$ and $k_i = 1$)

start round

choose x randomly uniformly from $\{1, 2, \dots, \beta_{kv}\}$ if $x \neq \beta_{kv}$ then do

$c-1$ no-ops; go to finish

repeat:cobegin (comment in parallel for $j = 1, \dots, k$)

$W_{ij} = 1$ (comment allocate resource)

if $E_{ji} = 1$ then $L_i = 0$ else $L_i = 1$

coend (comment end parallel)

if $L_i = 1$ AND $\text{counter}_i < c-u$ then

begin (comment start user's round)

cobegin (comment All resources are allocated)

$A_{ij} = c$ (comment accept)

use resources A_{ij} for u steps

$N_{ij} = 1$ (comment release resource)

coend (comment end parallel)

$N_{ij} = 0$; $A_{ij} = 0$

wait until $\text{counter}_i = c$

go to finish

end

else

begin

cobegin

$A_{ij} = 0$ (comment deny allocation)

coend

if $\text{counter}_i < c-u-2$ then go to repeat

else

begin

wait until $\text{counter}_i = c$

end

end

finish: end round

The resource allocator's round for allocator j.
 (Initially $M_j = 0$, $G_{ij} = 0$, $a_{ij} = 0$)

local a_{ij}

start round

In parallel do until counter_j = μ

if $W_{ij} = 1$ then ($M_j = M_j + 1$; $a_{ij} = 1$)

od

cobegin

if ($M_j = 0$ or $M_j > 2$) then

begin

$E_{ji} = 0$

go to finish

end

elseif

begin; if

if $a_{ij} = 1$ then

begin

$E_{ji} = 1$

if $A_{ij} = 0$ then begin

$E_{ji} = 0$

go to finish

end

else

begin

repeat until $N_{ij} = 1$

(COMMENT: resource allocated
until notification by user)

$E_{ji} = 0$

go to finish

end

end

else

repeat $E_{ji} = 0$ until resource deallocated

end

coend

finish: end round

Appendix II

II.1 The proof of the properties of the bidding algorithm for the case of equispeed processes.

Let us analyze first the simple case in which a fixed time instant t_0 exists after which all processes are executing at the same speed ($r_{\min} = r_{\max}$). However, for $t < t_0$ the processes may have been asynchronous and so, at $t > t_0$, the times of execution of their programs may be shifted relative to each other, in an adverse way, due to the oracle A. In this case, $\beta = 2$ is enough.

Proof of Proposition 1

Assume that user i has just been declared a winner in his current round because he had drawn the number βk_i in the probabilistic selection. Within a step, all resource allocators are notified. If some resource allocator is in an allocating period at the time of notification, then he is going to enter a monitoring period by at most a number of steps equal to the allocating period. If a resource allocator was in a monitoring period at the time of notification, then he will continue being in such a period (since his resource cannot be allocated to another winner, due to the presence of winner i). So, by at most a number of steps equal to an allocating period from the time i decided that he is a winner, all his requested resources are going to be notified, at the same time, within their monitoring period, that i is a winner.

Proof of Proposition 3

Since users' rounds are of equal time duration in the case of equispeed processes, only one draw of another user is possible within user i 's round. Actually, exactly one draw of each competing user will take place somewhere within user i 's round.

Proof of Proposition 2

The probability that user i remains a unique winner during his current round for all his wanted resources, given that i is a winner, is equal to the probability that none of the competing users manages to be a winner within i 's round. The number of the competing users is at

most kv (at most v competitors per each of the k resources asked by user i) and each competing user has exactly 2 draws which could make him a winner during i 's current round (by virtue of Proposition 3 above, one is the draw during the current i 's round and another is the draw during the previous round. The probability of each draw failing to win is $1 - \frac{1}{2kv}$, (since, in the case of equispeed processes, $\beta = 2$), hence the probability that i stays a unique winner during his current round is at least $(1 - \frac{1}{2kv})^{2kv} > \frac{1}{2e}$, where $e = 2.73$.

Proof of Theorem 1

The probability $p(\Gamma_t, A)$, for oracle A and history Γ_t , that a user i is going to be allocated all his wanted resources in his current round starting at t , never exceeds $\frac{1}{\beta kv}$ due to the fact that $\text{prob}\{\text{user } i \text{ chooses to be a winner in his current round}\} = \frac{1}{\beta kv}$. Given that user i chooses to be a winner, if he remains a unique winner during all of his current round, he is going to be allocated all of his resources (due to Proposition 1) with certainty. Multiplying probabilities given by Proposition 2, we get that

$$p(\Gamma_t, A) > \frac{1}{2kv} \frac{1}{2e} = \frac{1}{4ekv}$$

Hence $d = 4e$ for equispeed processes.

Proof of Theorem 2

Let u be the number of rounds required for user i to be granted all its k resources in some round, given that i starts requesting them at time t_1 and assuming that processes are equispeed after t_1 and also assuming any history of the system up to t_1 and any oracle A . Let round i start at time t_i $i < n$. We have by Baye's formula

$$\text{Prob}(u=m) = (1-p(\Gamma_{t_1}, A)) \cdots (1-p(\Gamma_{t_{m-1}}, A)) \cdot p(\Gamma_{t_m}, A)$$

By use of Theorem 1, we get

$$\text{Prob}(u=m) \leq (1 - \frac{1}{4ekv})^{m-1} \frac{1}{2kv}$$

If $u(\epsilon)$ is the least number such that

$$\text{prob}\{u > u(\epsilon)\} \leq \epsilon, \text{ then}$$

$$u(\epsilon) \leq \frac{\log(\frac{\epsilon}{2e})}{\log(1 - \frac{1}{4ekv})}$$

Since $\log(1 - \frac{1}{4ekv}) > -\frac{1}{4ekv}$ we have $u(\epsilon) \leq 4ekv \log(\frac{2e}{\epsilon})$.

Each allocation part of a resource's round takes $2+\mu$ steps. So, it is enough for the length of the user's round to be equal to $2(2+\mu)+1$, by the proof of Proposition 1. This implies that

duration of a user's round $\leq (2(2+\mu)+1)r_{\max}$ which implies (for μ independent of k, v)

$$\text{Prob}\{\gamma_{A,k} \leq (2(2+\mu)+1)r_{\max} \cdot u(\epsilon)\} \geq 1-\epsilon$$

i.e.

$$\gamma_k(\epsilon) = O(\mu kv \log(\frac{1}{\epsilon})) = O(kv \log(\frac{1}{\epsilon}))$$

and

$$\bar{\gamma}_k = O(kv).$$

II.2 The case of tame processes

In this case we no longer assume that the processes are equispeed, but that they are tame, as defined in Section 1.1. Our proof parallels that of the equispeed case of the previous paragraph. However, in the more general case of tame processes, we have to set $\beta = 2 \frac{r_{\max}}{r_{\min}}$, due to Proposition 3.

In Theorem 1, the constant d is now $4 \frac{r_{\max}}{r_{\min}}$ e. A proof of a generalization of Proposition 3 is easy to derive since durations of user's rounds are equal in terms of number of steps and a step of the slowest process can correspond to at most $r_{\max}/r_{\min}$ steps of the fastest process. Finally, the allocation duration now can be at most $(2+\mu)r_{\max}$ time, hence we have to allow for the length in steps of the user's round to be $(2(2+\mu)+1) \frac{r_{\max}}{r_{\min}}$ steps, leading to at most $(2(2+\mu)+1)(r_{\max}^2/r_{\min})$ time for the user's round. This leads to

$$\gamma_k(\epsilon) = O(\mu \frac{r_{\max}^2}{r_{\min}} kv \log(\frac{1}{\epsilon}))$$

implying again $\bar{\gamma}_k = O(kv)$.

NYU
Comp. Sci. Dept.
TR-62
Reif
Optimal distributed resource
allocation.

c.1

NYU
Comp. Sci. Dept.
TR-62 Reif

c.1

AUTHOR
Optimal distributed resource

TITLE
allocation.

DATE DUE

BORROWER'S NAME

This book may be kept

NOV 1 1983

FOURTEEN DAYS

A fine will be charged for each day the book is kept overtime.

